软件定义网络可信连接设计与实现 *

李兆斌，刘梦甜，魏占祯，王守融

(北京电子科技学院，北京 100070)

摘 要：软件定义网络（software defined networking，SDN）将控制层和数据转发层分离，由控制层对数据转发层进行统一管理。目前控制层及数据转发层设备间完整性认证机制尚不完善，若平台完整性损坏的设备接入网络，会给整个 SDN 网络带来严重的安全问题。为确保双方设备在完整可信的前提下建立连接，进而在源头上保障设备安全、网络可信，提出了一种新的 SDN 可信连接方案。该方案以可信网络远程设备认证技术为基础，利用可信平台模块作为可信支撑，在 SDN 数据转发设备与控制器的连接过程中添加完整性认证环节。测试分析表明，该方案有效可行，符合实际应用。

关键词：软件定义网络；可信连接；完整性认证；网络安全

中图分类号：TP309　　doi: 10.3969/j.issn.1001-3695.2017.09.0937

Design and realization of SDN trusted connection

Li Zhaobin, Liu Mengtian, Wei Zhanzhen, Wang Shourong

(*Beijing Electronic Science Technology Institute*, *Beijing* 100070, *China*)

Abstract: Software-Defined Networking separates the control layer and the data layer. Data forwarding is unified management by the control layer in SDN. However, equipment integrity authentication mechanism is not consummate between the control layer and the data layer. If the falsified equipment tries to connect the network, the whole network will face serious security problems. For ensuring that the connection was established after proving the equipment credible and integrated and that network is available, this paper proposed a project of trusted connection based on SDN. Combing the trusted network remote device authentication technology and using the trusted platform module as trusted support, the project added integrity certification to linking process of data forwarding devices and controllers. According to the experiment, the project is suitable for actual network environment.

Key Words: software defined networking; trusted connection; integrated authentication; network security

0　引言

软件定义网络（software defined networking，SDN）将控制层和数据转发层分离后，由控制层通过接口对数据转发层进行统一连接及管理，以便简化网络管理，提高网络灵活性，降低网络调整配置成本。然而，更强的可变性在带来优势的同时也产生了一些亟需解决的安全问题。作为各类网络功能实现的前提，如何使控制层和数据转发层中的设备正常且安全建立连接就是其中之一。

当数据转发层的数据转发设备（open vSwtich, OVS）新加入网络时，可通过配置位于控制层的控制器 IP 地址建立底层通信连接，并在此基础上开始协商双方共同支持的 OpenFlow 协议最高版本，若协商顺利，连接建立成功。也就是说，控制器与 OVS 建立连接时并未对即将连接的对端设备进行平台完整性认证，两类设备都有可能与被窜改后的设备进行连接。控制器作为 SDN 的核心，一旦遭受攻击而无法正常工作，可能会导致网络的瘫痪，而 OVS 作为具体工作的执行者，若被危险设备利用，同样会对网络报文转发甚至对整体网络造成影响[1,2]。因此，在连接建立前添加设备完整性校验对解决控制器与 OVS 连接认证问题极为必要。

在认证连接这一问题上， OpenFlow 规范允许用户在通道建立时自主选择是否采用 TLS 安全机制。但 TLS 在实际部署中的复杂性及自身局限性导致安全通道实现困难，因此大多数用户还是更倾向于直接采用 TCP 明文来建立连接通道。况且，TLS 的保护目标是数据，并不能对设备的软、硬件完整性校验提供帮助[3]。

基金项目：国家重点研发计划项目（2017YFGX110123）；中央高校基本科研业务专项资金项目（2017CL04）；北京市自然科学基金资助项目（4152048）

作者简介：李兆斌（1977-），男（蒙古族），内蒙古人，副研究员，博士，主要研究方向为软件定义网络、网络安全与测评（bestibesti@163.com）；刘梦甜（1992-），女，硕士研究生，主要研究方向为软件定义网络、网络安全与测评；魏占祯（1971-），男，教授，硕士，主要研究方向为软件定义网络、网络安全与测评、可信网络；王守融（1992-），男，硕士研究生，主要研究方向为软件定义网络、网络安全与测评.

另外，通过引入第三方认证中心为设备颁发证书也是一种认证设备的方案，但因处理方式复杂，并没有被广泛应用。除此之外，物理隔离也被视为保障控制器及其他设备安全的有效方法，不过该方法对部署环境、管理员管理等多方面的要求很高，并不容易实现[3]。张团利等人[4]提出控制器与交换机失效后的路径保护、链路保护等恢复机制，虽然可以有效帮助遭受攻击的设备在短时间内恢复功能，但已连接的仿冒设备若重复发起攻击，仍会对网络造成影响。而周睿康[5]、潘秋月[6] 也仅仅是分别引入可信计算技术完成控制器间的认证或对交换机协议方面入手进行改进。

不难发现，现有设备认证方案鲜有提及控制器与交换机间认证，然而建立安全连接作为一切功能的起点对整体网络运行至关重要，所以完善连接过程的必要性不言而喻。经过分析，本文结合可信网络远程证明协议，利用可信平台模块作为可信支撑，选择 Ryu 作为控制器，在不改变流程的前提下添加设备完整性认证过程，对 OpenFlow 协议、OVS 系统及 Ryu 系统进行改造扩展，实现 Ryu 与 OVS 可信连接，避免篡改后设备威胁网络安全。

1　相关工作

1.1　SDN 与 OpenFlow

SDN 架构[7]如图 1 所示，分别是应用层、控制层以及数据转发层。控制层与另外两层分别通过北向接口及南向接口（如 OpenFlow）进行数据交互。控制层根据应用层功能需求调用数据转发层实现功能，而数据转发层则根据控制层所发出的指令进行数据转发、存储、上传等工作。

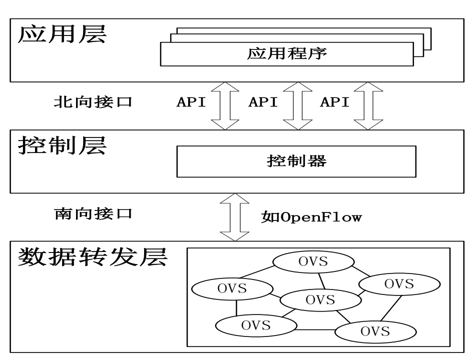


图 1　软件定义网络基本架构

OpenFlow 作为构建 SDN 网络的标准规范，是一种集中控制模型，其要素包含控制器、OVS、两者间 OpenFlow 通道建立的连接以及 OVS 流表[8]。在 SDN 中通道建立、设备配置、定期交互实现均通过 OpenFlow 消息来实现。OpenFlow 支持以下三类消息：Controller-to-Switch（控制器到数据转发设备消息）、Asynchronous（异步消息）以及 Symetic（对称消息）[8]。其中，Symetic 是控制器和数据转发设备都可以自主发起的，一般用来进行设备初始化，在发送前也无须进行请求和确认。在控制器和数据转发设备建立连接前发送的 Hello 消息以及定期发送的 ECHO 信息都是属于 Symetic，该类消息虽然是最简单的一种，但对于通道建立以及维护稳定连接来说不可或缺。

1.2　可信平台与远程设备可信认证

可信平台中身份认证密钥（attestation identity key,AIK）用于提供平台的身份证明[9]。平台配置寄存器（PCR）则通过记录动态可信度量摘要值来记录系统运行状态[10]，因为 PCR 值对于设备完整性认证至关重要，所以为避免被恶意窜改或伪造，寄存器不能通过端口随意读写，仅能通过重置操作和扩展操作来变更。

可信计算认证技术利用内嵌硬件设备 TPM 加密芯片在设备启动时执行不可逆的信任传递机制，利用一环度量一环的基本原则完成自系统固件至设备平台的度量，并将各阶段可信度量摘要值存储至 PCR 中[11~12]，通过对比 PCR 值来识别设备软硬件是否经过窜改，而非仅依赖于系统启动完成后的用户身份验证。

不过，如果涉及到远程设备身份认证时，可信认证技术需要配合远程证明技术共同完成。

目前，匿名认证方案（direct anonymous authentication, DAA）作为最基础的远程证明方案被广泛应用在远程身份设备认证中。该方案在交互双方中确定不同可信域的质询方和证明方后，通过挑战/应答协议交换并判断认证值的方式完成认证[13]。

借助远程认证技术，可信平台也可以对远程设备进行设备完整性报告。远程质询方（挑战者）首先利用产生随机数并发送给证明方（应答者），证明方利用认证身份密钥对当前设备的 PCR 值以及随机数一同进行签名后报告给质询方，质询方可利用随机数来避免重放攻击，利用证明方 PCR 值与期望值对比可证明其可完整可信（图 2）。

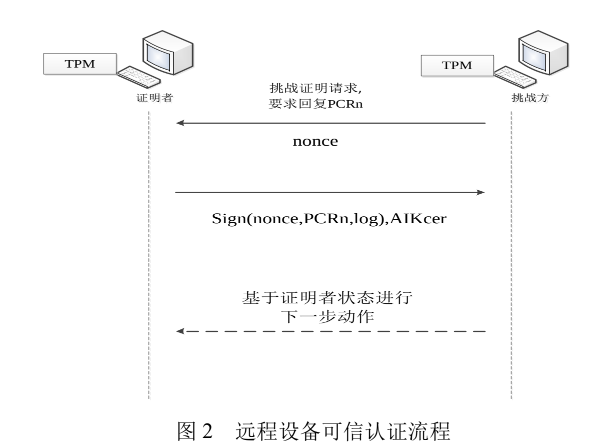


图 2　远程设备可信认证流程

2　SDN 可信连接设计

2.1　可信连接模型设计

SDN 可信连接模型跨越控制层与数据转发层，主要由可信平台模块、控制器、OVS 以及负责完整性认证的可信连接通道组成。模型中的 OVS 与控制器在建立连接时通过验证对方平台完整性校验值来确认其设备是否可信，从而决定是否完成本次连接的建立。只有双方同时认证通过，连接才可成功建立并继续进行后续的配置查询交互，有一方认证失败，则停止连接。

可信连接模型如图 3 所示。

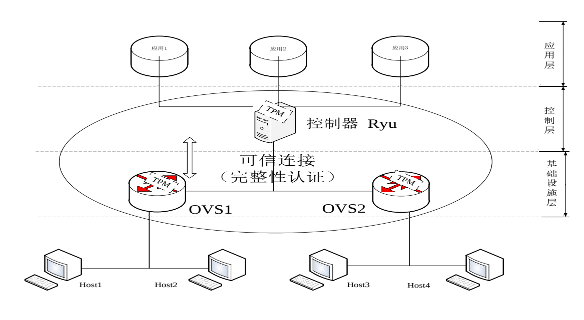


图 3　可信连接模型

2.2　完整性认证过程概述及安全性证明

2.2.1 具体完整性认证过程

完整性认证过程为 SDN 可信连接模型中重要部分，具体过程如图 4 所示。完整性认证过程符号解释说明如表 1 所示。

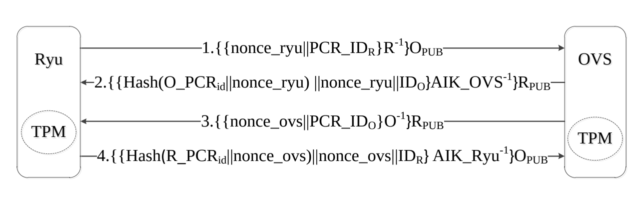


图 4　可信认证过程

表 1　完整性认证过程符号解释说明

R	控制器，认证主体之一
O	交换机，认证主体之一
nonce_ryu	Ryu 请求认证时产生的随机数，与交换机相应度量值进行散列运算
nonce_ovs	OVS 请求认证时产生的随机数，与控制器相应度量值进行散列运算
PCR_ID_R	Ryu 请求认证时指定认证的 PCR 序号，OVS 需根据序号提供特定的值进行认证
PCR_ID_O	OVS 请求认证时指定认证的 PCR 序号，Ryu 需根据序号提供特定的值进行认证
O_{PUB}	OVS 预置公钥
R_{PUB}	Ryu 预置公钥
O^{-1}	OVS 预置私钥
R^{-1}	Ryu 预置私钥
R_PCR	Ryu 指定序号的 PCR 值
O_PCR	OVS 指定序号的 PCR 值
ID_R	Ryu 设备平台 ID
ID_O	OVS 设备平台 ID
AIK_OVS^{-1}	OVS 身份认证密钥私钥，用于校验数据签名
AIK_Ryu^{-1}	Ryu 身份认证密钥私钥，用于校验数据签名
Hash( )	TPM 中基于 SHA-1 引擎的 Hash( )函数

2.2.2 完整性认证过程形式化描述以及安全性证明

BAN 逻辑[5]是多种证明协议安全性逻辑系统中最具影响力工具，由 10 个基本语法语义组成，另外 BAN 逻辑推理还包括由 7 个定理组成的 5 个主要规则。下面将以 BAN 逻辑对完整性认证过程进行安全性证明。

1）完整性认证过程形式化描述

$R \rightarrow O$ ：$\{\{nonce_ryu||PCR_ID_R\}R^{-1}\}O_{PUB}$

$O \rightarrow R$ ：$\{\{Hash(O_PCR_{id}||nonce_ryu)\ ||nonce_ryu||ID_O\}AIK_OVS^{-1}\}R_{PUB}$

$O \rightarrow R$ ：$\{\{nonce_ovs||PCR_ID_O\}O^{-1}\}R_{PUB}$

$R \rightarrow O$ ：$\{\{Hash(R_PCR_{id}||nonce_ovs)\ ||nonce_ovs||ID_R\}AIK_Ryu^{-1}\}O_{PUB}$

2）BAN 逻辑安全目标

根据完整性认证过程的安全需求，设定安全目标如表 2 所示。

表 2　完整性认证过程安全目标设定

$O\ |\equiv R\ |\equiv nonce_ryu$	OVS 相信 Ryu 相信 nonce_ryu 可信
$R\ |\equiv O\ |\equiv nonce_ovs$	Ryu 相信 OVS 相信 nonce_ovs 可信
$O\ |\equiv R\ |\equiv PCR_ID_R$	OVS 相信 Ryu 相信 PCR_ID_R 可信
$R\ |\equiv O\ |\equiv PCR_ID_O$	Ryu 相信 OVS 相信 PCR_ID_O 可信
$O\ |\equiv R\ |\equiv R_PCR_{id}$	OVS 相信 Ryu 相信 nonce_ryu 可信
$R\ |\equiv O\ |\equiv O_PCR_{id}$	Ryu 相信 OVS 相信 nonce_ovs 可信
$O\ |\equiv R\ |\equiv ID_R$	OVS 相信 Ryu 相信 ID_R 可信
$R\ |\equiv O\ |\equiv ID_O$	Ryu 相信 OVS 相信 ID_O 可信

3）BAN 逻辑初始化假设

为验证上述 BAN 逻辑安全目标，对其进行符合 BAN 逻辑的初始化假设：

① $O\ |\equiv \xrightarrow{AIK_Ryu} R$　　$R\ |\equiv \xrightarrow{AIK_OVS} O$

② $O\ |\equiv \xrightarrow{R_{PUB}} R$　　$R\ |\equiv \xrightarrow{O_{PUB}} O$

③ $O\ |\equiv \#(nonce_ryu)$　　$R\ |\equiv \#(nonce_ovs)$

④ $O\ |\equiv \#(PCR_ID_R)$　　$R\ |\equiv \#(PCR_ID_O)$

⑤ $O\ |\equiv \#(ID_R)$　　$R\ |\equiv \#(ID_O)$

⑥ $O\ |\equiv \#(R_PCR_{id})$　　$R\ |\equiv \#(O_PCR_{id})$

4）BAN 逻辑安全证明

利用 3）中安全初始化假设，对 2）中安全目标进行推理验证：

①推理 1 由 $R \rightarrow O$ ：$\{\{nonce_ryu||PCR_ID_R\}R^{-1}\}O_{PUB}$

因 $O\ |\equiv \xrightarrow{R_{PUB}} R$，$O \triangleleft \{nonce_ryu||PCR_ID_R\}R^{-1}$

根据消息含义规则[5]可推：$O|\equiv R|\sim nonce_ryu||PCR_ID_R$

根据接收消息规则以及信念规则[5]可推：

$O|\equiv R|\sim nonce_ryu$　$O|\equiv R|\sim PCR_ID_R$

因 $O\ |\equiv \#(nonce_ryu)$ 且 $O\ |\equiv \#(R_PCR_{id})$

根据临时值验证规则[5]可得

$O\ |\equiv R\ |\equiv nonce_ryu$　$O\ |\equiv R\ |\equiv PCR_ID_R$

②推理 2 由 $O \rightarrow R$：$\{\{Hash(O_PCR_{id}||nonce_ryu)\ ||nonce_ryu||ID_O\}AIK_OVS^{-1}\}R_{PUB}$

因 $R\ |\equiv \xrightarrow{AIK_OVS} O$，

$R \triangleleft \{Hash(O_PCR_{id}||nonce_ryu)||nonce_ryu||ID_O\}AIK_OVS^{-1}$

根据消息含义规则[5]可推：

$R|\equiv O|\sim Hash(O_PCR_{\mathrm{id}}||nonce_ryu)||nonce_ryu||ID_O$

根据接收消息规则与信念规则可推：

$R|\equiv O|\sim O_PCR_{\mathrm{id}}$　$R|\equiv O|\sim ID_O$

因$R|\equiv \#(O_PCR_{\mathrm{id}})$ 且 $R|\equiv \#(ID_O)$

根据临时值验证规则可得

$R|\equiv O|\equiv O_PCR_{id}$　$R|\equiv O|\equiv ID_O$

③推理 3 由 O → R ：$\{\{nonce_ovs||PCR_ID_O\}O^{-1}\}R_{PUB}$ 可知与推理 1 同理，从而可得

$R|\equiv O|\equiv nonce_ovs$　$R|\equiv O|\equiv PCR_ID_O$

④推理 4 由 R → O ：$\{\{Hash(R_PCR_{id}||nonce_ovs)||nonce_ovs||ID_R\}AIK_Ryu^{-1}\}O_{PUB}$ 可知与推理 2 同理，从而可得：$O|\equiv R|\equiv R_PCR_{id}$　$O|\equiv R|\equiv ID_R$。

从上述 BAN 逻辑推理可以看出，结果与 2）所设置的安全目标一致，该过程可以保证随机数、PCR 值等信息的安全性，因此完整性认证过程可以满足可信连接模型的安全需求。

2.3　可信连接建立过程（图 5）

① OVS 与 Ryu 进行连接初始化并开始建立连接；

② 发送 OFPT_HELLO 消息开始协商双方共同支持的最高 OpenFlow 协议版本；

③ Ryu 发起认证请求，发送 OFPT_AUTH_REQUEST 消息（含{{nonce_ryu||PCR_IDR}R-1}OPUB）；

④ OVS 响应 Ryu 认证请求，回复 OFPT_AUTH_COMPARE 消息（含{{Hash(O_PCRid||nonce_ryu)||nonce_ryu||IDO}

AIK_OVS-1}RPUB）；

OVS 发起认证请求，发送 OFPT_AUTH_REQUEST 消息（含{{nonce_ovs||PCR_IDO}O-1}RPUB）；

⑤ Ryu 响应 OVS 认证请求，回复 OFPT_AUTH_COMPARE 消息（含{{Hash(R_PCRid||nonce_ovs)||nonce_ovs||IDR}

AIK_Ryu-1}OPUB）；

⑥ Ryu 对 OVS 进行认证。若认证失败，连接建立失败，回复认证失败 OFPT_ERROR 信息；若认证成功，连接建立成功，继续发送后续 OFPT_FEATURES_REQUEST 消息，查询 OVS 配置；

⑦ OVS 对 Ryu 进行认证。若认证失败，连接建立失败，直接丢弃连接；若认证成功，则将当前连接状态设置为成功，并继续接收、回复 OFPT_FEATURES_REPLY 消息；

⑧ 定期交互 FPT_ECHO_REQUEST/OFPT_ECHO_REPLY 消息，确认连接状态。

SDN 可信连接实现。流程图如图 5 所示。

2.4　OVS 与 Ryu 连接建立过程

OVS 可通过配置 Ryu 地址与其建立底层连接，此时并无 OpenFlow 协议消息交互。OVS 系统开始运行后会通过连接管理实体识别底层连接是否建立完成。若已建立，则向 Ryu 发送 OPFT_HELLO 消息用于协商协议版本，协商完成即表示 OpenFlow 通道建立完成。OVS 具体程序执行过程如图 6 所示。

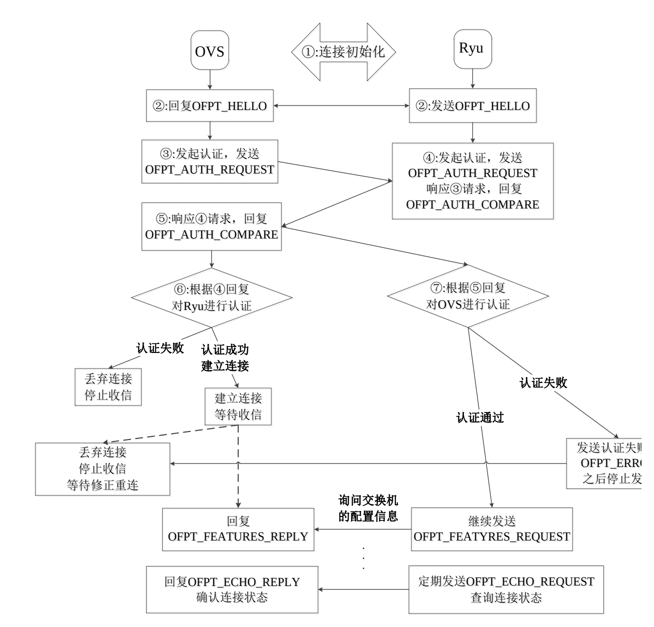


图 5　SDN 可信连接建立流程

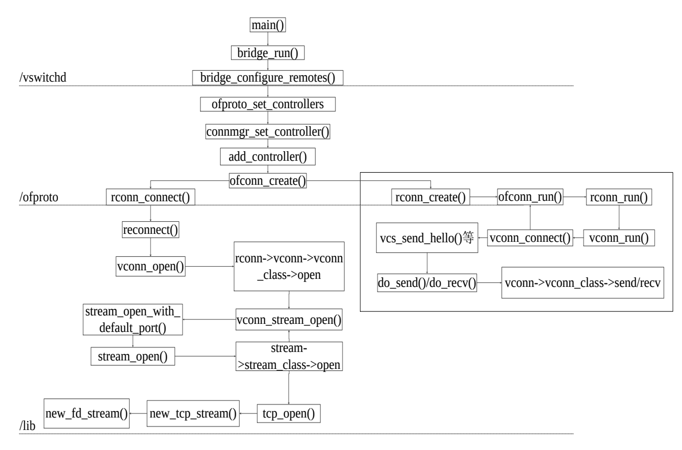


图 6　OVS 连接建立程序执行流程

OVS 中 ofconn、rconn、vconn 三类实体会对连接进行逐级管理。vconn 作为最基础的连接管理实体，最主要的工作就是负责 OFPT_HELLO 消息的发送和接收，并可快速将当前连接情况向上报告（图 6 标示部分），rconn 以及 ofconn 将根据 vconn 传递的结果执行后续动作。

Ryu 并非如 OVS 一般复杂。启动后，Ryu 开始接收 OVS 消息并进行回复，版本协商一致后则确认连接建立并进行配置询问。

因此，SDN 可信连接实现可分别从 OVS 三类连接管理实体以及 Ryu 连接响应部分入手。

2.5　SDN 可信连接实现

2.5.1 OpenFlow 协议可信连接实现

1）添加 OpenFlow 可信连接消息类型

① 完整性认证请求消息（OFPT_AUTH_REQUEST）

如图 7 所示，消息来源以及数据内容各 8 位，分别用于表示该消息发送源以及所携带数据内容；数据为 160 字节，用于

发送签名并加密后的随机数 nonce_ryu 或 nonce_ovs 以及指定的 PCR 序号。

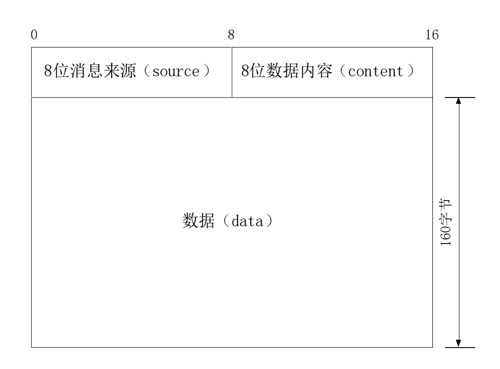


图 7　可信连接认证请求消息结构

表 3　可信连接认证请求消息说明

source	0000	来源于 OVS
	0001	来源于 Ryu
content	0000	数据内容为随机数
	0001	数据内容为其他（供扩展用）

综合考虑各版本协议消息情况，将认证请求消息识别码设定为 40。

② 完整性校验值回复消息（OFPT_AUTH_COMPARE）

如图 8 所示，消息来源以及数据置入方式各 8 位，分别用于表示该消息发送源以及所携带数据内容置入系统方式；数据为 160 字节经加密及签名后的完整性校验值

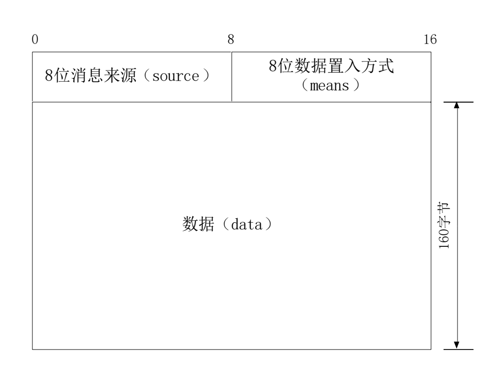


图 8　可信连接认证值回复消息结构

表 4　可信连接认证值回复消息说明

source	0000	来源于 OVS
	0001	来源于 Ryu
means	0000	数据为自动置入
	0001	数据为管理员手动置入

综合考虑各版本协议消息情况，将完整性校验值回复消息识别码设定为 41。

2）添加错误消息类型——认证失败

当交换机身份并未通过控制器认证时，控制器下发错误类型为：认证失败（AUTH_FAILED）的错误消息，提示待连接的交换机当前认证及连接建立状态。该环节没有直接由控制器丢弃连接，而是先发出错误消息，告知认证失败原因。当控制器被交换机视为不可信设备时，交换机不考虑其认证失败原因，直接丢弃连接停止继续收信。

2.5.2 Ryu 系统可信连接实现

1）添加认证状态调度器

在 Ryu 核心程序 handler.py 中 handshake 以及 config 两个调度器之间添加认证状态调度器，在 ofp_handler.py 等执行具体函数的文件中添加相应调度器，以保证相关函数顺利运行。

Ryu 中原有 handshake\config\dead\main 四个调度器，其中 handshake 与 config 分别承担连接建立前 Hello 握手以及建立后配置的工作，无论是发送、接收 Hello 消息还是发送配置询问消息，相应函数执行前都需先将状态进行调整。

2）定义各项新添变量

在 ofproto_v1_0.py、ofproto_v1_2.py、ofproto_v1_3.py、ofproto_v1_4.py 等各版本文件中定义新添 OpenFlow 消息识别码、消息格式、消息中来源等变量识别码以及认证失败错误消息类型识别码等变量。

3）构造、发送、接收认证请求消息及设备完整性验证值回复消息

当前版本 Ryu 可以支持多版本 OpenFlow 协议，以 OpenFlow_1.3 为例说明本部分实现。

在 ofp_handler.py 及 ofproto_v1_3_parser.py 中共同实现：

①认证请求消息构造与解析

```
self.logger.debug('move onto authentication')
datapath.set_state(AUTHENTICATION_DISPATCHER)
auth_request_msg = datapath.ofproto_parser.OFPAuthenticationRequest(datapath)
```

```
@_register_parser
@_set_msg_type(ofproto.OFPT_AUTHENTICATION_REQUEST)
class OFPAuthenticationRequest(MsgBase):
    def __init__(self,datapath, source=None, content=None, data=None):
    @classmethod
    def parser(cls, datapath, version, msg_type, msg_len, xid, buf):
    def _serialize_body(self):
```

②设备完整性验证值回复消息构造与解析

```
@set_ev_handler(ofp_event.EventOFPAuthenticationRequest, AUTHENTICATION_DISPATCHER)
def switch_authentication_request_handler(self,ev):
```

```
@_register_parser
@_set_msg_type(ofproto.OFPT_AUTHENTICATION_COMPARE)
class OFPAuthenticationCompare(MsgBase):
    def __init__(self,datapath,source=None, means=None, data=None):
    @classmethod
    def parser(cls, datapath, version, msg_type, msg_len, xid, buf):
    def _serialize_body(self):
```

③证明方完整性校验

```
@set_ev_handler(ofp_event.EventOFPAuthenticationCompare, AUTHENTICATION_DISPATCHER)
def switch_authentication_compare_handler(self,ev):
```

如校验成功，则继续进行后续配置询问流程；若失败，则发送认证失败错误消息。

2.5.3 OVS 系统可信连接实现

1）在底层连接控制实体中增添“认证”状态。

在 vconn_state、vconn_run、vconn_run_wait 等环节中添加认证请求消息发送 VCS_SEND_RYU_AUTH_REQUEST、认证请求消息接收 VCS_RECV_RYU_AUTH_REQUEST、校验值回复信息发送 VCS_SEND_RYU_AUTH_COMPARE、校验值回复

信息接收 VCS_RECV_RYU_AUTH_COMPARE 四类状态，保证消息构造发送、接收解析能够顺利执行。

2）定义新消息结构

```
/* OFPT_AUTHENTICATION_REQUEST: Authentication request message (controller -> datapath). */
struct ofp_authentication_request {
OFP_ASSERT(sizeof(struct ofp_authentication_request) == 116);

/* OFPT_AUTHENTICATION_COMPARE: Authentication value message ( datapath -> controller). */
struct ofp_authentication_compare {
OFP_ASSERT (sizeof(struct ofp_authentication_compare) == 184);
```

为保证 OpenFlow 消息处理函数顺利解析各类消息，在 ofp-util.h 中同样定义新消息结构：

```
struct ofputil_auth_request
{
struct ofputil_auth_compare
{
```

3）编写消息接收、提取接口

可信连接在连接初始阶段实现，此时并未进入 OpenFlow 协议消息处理环节中，消息处理略有不同。系统原有消息提取接口不能在本阶段从认证请求消息以及完整性校验值回复消息中准确提取所需信息，同时为避免和其他交互信息发生冲突，因此在 ofpbuf.h 中编写消息提取接口 ofpbuf_auth_pull：

```
static inline void *ofpbuf_auth_pull(struct ofpbuf *b, size_t size)
{
```

4）消息构造、发送与接收

在 vconn.c 与 ofp-util.c 两个文件中编写承担主要工作的程序。

① 实现认证消息构造与发送：

```
struct ofpbuf *
ofputil_encode_auth_request(const struct ofputil_auth_request *request, enum ofp_version version,ovs_be32 xid)
{
```

② 实现认证消息接收与解析：

```
static void
vcs_recv_ryu_auth_request(struct vconn *vconn)
{
```

③ 实现设备完整性校验值回复消息构造与发送：

```
struct ofpbuf *
ofputil_encode_auth_compare(const struct ofputil_auth_compare *compare, enum ofp_version version,ovs_be32 xid)
{
```

④ 实现设备完整性校验值回复消息接收与完整性校验：

```
static void
vcs_recv_ryu_auth_compare(struct vconn *vconn)
{
```

```
bool
decode_auth_compare(struct ofputil_auth_compare *compare,const struct ofp_header *oh)
{
```

5）在上层调用程序中完善认证消息处理分支

分别在 rconn.c、ofproto.c 等相关文件中，完善认证消息处理分支，以避免系统处理 OpenFlow 消息时将认证相关消息视为错误信息而影响完整性认证及后续连接建立过程。

```
switch (type) {
case OFPTYPE_HELLO:
case OFPTYPE_ERROR:
case OFPTYPE_AUTHENTICATION_REQUEST:
case OFPTYPE_AUTHENTICATION_COMPARE:
```

6）完善 ofp-errors.h\ofp-errors.c\ofp-error.inc、ofp-msgs.h\ofp-msgs.c\ofp-msgs.inc，编写 ofp-auth.h\ofp-auth.inc

在 ofp-errors 相关文件中添加认证失败错误类型的识别码、失败原因、原因说明及识别码；在 ofp-msgs 相关文件中添加新消息类型名称、支持版本范围等基础内容；编写 ofp-auth 相关文件以说明完整性认证环节支持的版本情况。

3　SDN 可信连接测试

3.1　SDN 可信连接测试环境

3.1.1 可信平台模块仿真环境搭建

本文可信连接测试环境采用可信平台模块仿真器 TPM_emulator 代替 TPM 加密芯片完成设备平台软、硬件完整性的测试。TPM_emulator 由德国研究人员开发，是目前应用最广泛的 TPM 仿真器[14]，该软件使用 Cmake 组织源代码，可在 Linux 平台下使用，能够很好的模拟 TPM 的功能，包括随机数发生、生成 RSA 密钥以及进行数字签名等。

TPM_emulator 的运行需要依赖跨平台编译工具、运算库、图形库以及 TSS 软件栈、界面管理模块等，图形化管理界面做为选择性安装的模块可以帮助用户更好的管理可信平台。

3.1.2 SDN 可信连接测试拓扑搭建

SDN 可信连接网络拓扑主要由两个控制器以及三个交换机组成。其中 Ryu1、OVS1 以及 OVS3 为可信设备，OVS2 以及 Ryu2 为不可信设备。逻辑上，OVS1 以及 OVS2 直接连接 Ryu1，OVS3 则需要从 Ryu1 以及 Ryu2 中选择可信设备连接，实验设置其优先连接 Ryu2。只有双方同时通过可信认证，连接才可建立成功。

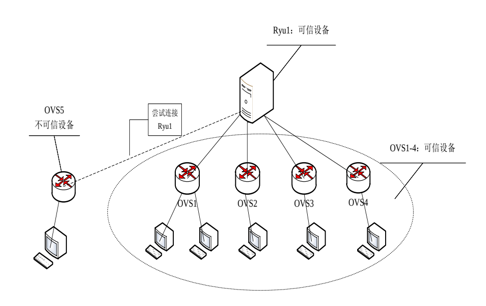


图 9　可信连接测试拓扑 1：控制器为可信控制器，测试 OVS

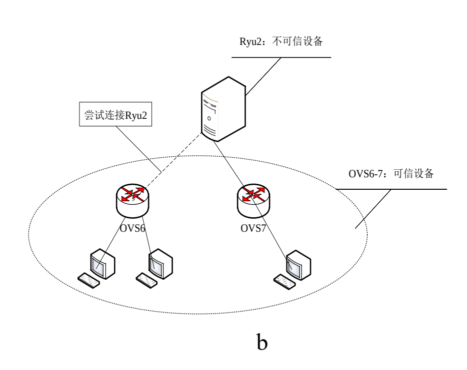
3.1.3 Wireshark 协同 OpenFlow 协议功能扩展

目前，Wireshark 较多版本已可以支持抓取、解析 OpenFlow 消息，但是自定义添加消息只能被抓取而不能作出解析，这样无疑是对可信连接后期分析过程造成了阻碍。因此，根据可信连接各模块的分析需求，对 Wireshark-2.0.9 中 OpenFlow 协议进行功能扩展，使其支持可信连接信息的解析，从而在测试阶段直观地对分析认证过程，对通信实体的认证情况进行把控。

3.1.4 SDN 测试环境软、硬件说明

改造工作主要包含以下四个方面：a）添加可解析的消息类型：进行完整性认证请求消息及完整性校验值回复消息的名称和消息编号的定义和匹配，实现抓包时的正常识别；b）定义并初始化消息中出现的各类变量：单独定义消息以及认证失败的

错误消息结构中各变量，并将其进行初始化，实现内容的顺利提取；c）编写自定义消息解析函数：配合已定义的变量，根据消息格式编写消息解析函数，实现消息的完整解析；d）将新添加消息的解析过程添加流程：在主函数中添加新消息处理的分支，实现对消息的识别与处理。



b

图 10　可信连接测试拓扑 2：控制器为不可信控制器，测试控制器

表 5　SDN 测试环境网络配置说明

设备名称	IP 配置	测试角色
Ryu1	192.168.2.158	可信控制器
OVS1	192.168.2.40	可信交换机，连接 Ryu1
OVS5	192.168.2.50	不可信交换机，连接 Ryu1
Ryu2	192.168.2.168	不可信控制器
OVS6	192.168.2.60	可信交换机,连接 Ryu2

表 6　SDN 测试环境硬件设备说明

设备名称	操作系统	功能
Ryu 1	Ubuntu14.04.1	SDN 控制器
Ryu 2	Ubuntu14.04.1	SDN 控制器
OVS 1	Ubuntu14.04.1	SDN 交换机
OVS 5	Ubuntu14.04.1	SDN 交换机
OVS 6	Ubuntu14.04.1	SDN 交换机

表 7　SDN 测试环境软件安装说明

软件名称及版本	软件功能	安装位置
Ryu	本文改造控制器	Ryu1 Ryu2
OVS-2.4.0	本文改造交换机	OVS1 OVS5 OVS6
Wireshark-2.0.9（扩展）	解析连接建立过程的交互信息	Ryu1 Ryu2
tpm_emulator 0.7.4 libgtk 2.0 TrouSerS 0.3.8 tpm_tools 1.3.8 tpmmanager 0.8.1	搭建可信平台仿真环境，实现可信认证	Ryu1 Ryu2 OVS1 OVS5 OVS6

3.2　测试步骤及结果分析

3.2.1 SDN 可信连接设备处理动作测试步骤

本测试将拓扑图中的设备分为可信控制器 Ryu1—可信交换机 OVS1、可信控制器 Ryu1—不可信交换机 OVS5、不可信控制器 Ryu2—可信交换机 OVS6 三组，旨在对交换机完整性认证设备处理动作、控制器完整性认证设备处理动作及各测试组交互信息数量情况进行测试。

(1)可信平台模拟器运行测试；

(2)测试交换机可信认证情况，启动可信控制器 Ryu1。

① 测试可信交换机 OVS1。

启动 OVS1，配置控制器 Ryu1 地址用于发现 Ryu1 ；观察 Ryu1 认证结果，利用 Wireshark 工具抓取认证及后续过程设备间交互信息。

② 测试不可信交换机 OVS5。

启动 OVS5，配置控制器 Ryu1 地址用于发现 Ryu1；观察 Ryu1 认证结果，利用 Wireshark 工具抓取、记录认证及后续过程设备间交互信息数据包。

(3)测试控制器可信认证情况，启动不可信控制器 Ryu2。

启动可信交换机 OVS5，配置控制器 Ryu2 地址用于发现 Ryu1；利用 Wireshark 工具抓取、记录认证及后续过程设备间交互信息数据包，观察 OVS6 认证后的动作响应。

(4)对三组测试数据包交互情况进行对比分析。

3.2.2 SDN 可信连接用时测试步骤

(1)可信平台模拟器运行测试

(2)启动改造后可信控制器 Ryu1，启动 OVS1，配置控制器 Ryu1 地址，记录可信连接建立用时。

(3)测试传统连接用时并与可信连接建立用时进行比较

3.2.3 可信平台模拟器运行测试结果

TPM 模拟环境成功运行，可以查看平台版本号、公钥等内容且 TPM_emulator 管理器启动成功。

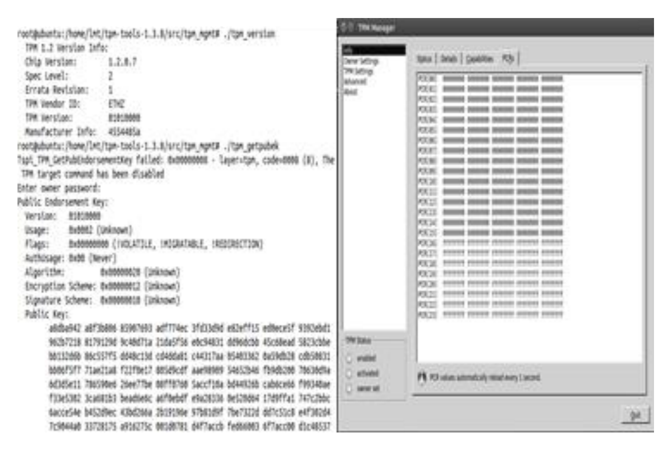


图 11　TPM_emulator 及界面管理器启动成功

3.2.4 SDN 可信连接设备处理动作测试结果及分析

SDN 可信连接设备处理动作 3 组测试结果如表 8 所示。

表 8　SDN 可信连接测试结果

认证组别	质询方	证明方	认证结果	设备处理动作
1	Ryu1	OVS1	成功	继续后续配置查询
	OVS1	Ryu1	成功	定期发送 OFPT_ECHO 消息探查连接情况
2	Ryu1	OVS5	失败	发送认证失败信息停止发包、丢弃连接（图 12）
	OVS5	Ryu1	成功	仅单方面建立连接控制器不响应其消息
3	Ryu2	OVS6	成功	控制器单方面连接交换机不响应其信息
	OVS3	Ryu2	失败	停止收信、丢弃连接

```
root@ubuntu:/home/lmt/ryu# ryu run
loading app ryu.controller.ofp_handler
instantiating app ryu.controller.ofp_handler of OFPHandler
authentication failed
```

图 12　Ryu1 认证 OVS5 结果

3 组测试结果及相应设备处理动作与可信连接流程设定一致，可见 SDN 可信连接能够完成连接时的设备完整性认证工作。

另外，测试组交互信息数量情况如图 13 所示。

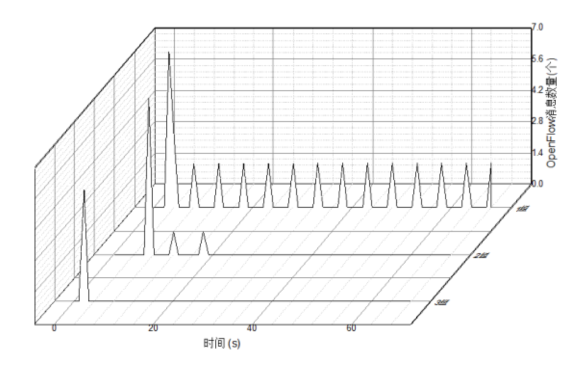


图 13　测试组 OpenFlow 信息交互情况

1 组认证结束后连接建立成功，并定期交互 echo 信息维持连接；2 组因认证失败没有后续配置信息的交互且连接丢弃，因此在后面的时间里 2 组不在出现 OpenFlow 消息；3 组中 OVS6 尝试连接 Ryu2，但因其不可信而断掉连接，后续并无信息交互。

3.2.5 SDN 可信连接用时测试结果及分析

与传统连接流程相比，加入可信连接过程后步骤有所增加，连接占用时间也略微产生变化，因此对添加可信连接过程与传统连接过程用时进行测试。

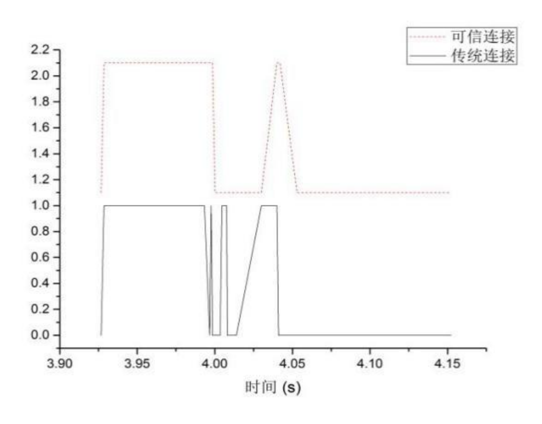


图 14　可信连接与传统连接建立完成时间对比结果

由测试结果（图 14）可见，虽然可信连接时间有所增加，但与传统连接时间相比，尚在可以接受的范围之内且没有对后续流程产生影响。

4　结束语

软件定义网络可信连接通过分析控制层与数据转发层间安全需求，确定控制层与数据转发层间完整性认证机制不完善是当前亟需解决的安全问题之一。经测试，本文提出的可信连接实现了在设备连接前完成设备完整性认证，可以有效防止遭窜改的设备恶意连接。不过，软件定义网络发展迅速，系统也在不断更新，除控制器与交换机间设备完整性认证问题外，应该继续选择适用、成熟的安全技术解决其他待解决的认证问题。

参考文献：

[1] Shu Z, Wan J, Li D, et al. Security in software-defined networking: threats and countermeasures [J]. Mobile Networks & Applications, 2016, 21 (5): 1-13.

[2] Bawany N Z, Shamsi J A, Salah K. DDoS attack detection and mitigation using SDN: methods, practices, and solutions [J]. Arabian Journal for Science & Engineering, 2017, 42 (2): 425-441.

[3] 左青云，张海粟. 基于 OpenFlow 的 SDN 网络安全分析与研究 [J]. 信息网络安全, 2015 (2): 26-32.

[4] 张国利，吕光宏，杨沛霖. 基于 OpenFlow 的 SDN 可靠性综述 [J]. 电子科技, 2016, 29 (2): 177-181.

[5] 周睿康. 基于 SDN 的可信网络系统研究 [D]. 北京：北京工业大学, 2015.

[6] 潘秋月. 基于 Open vSwitch 的可信交换机 STP 协议的可信改进 [D]. 北京：北京工业大学, 2014.

[7] 张朝昆，崔勇，唐翯祎，等. 软件定义网络 (SDN) 研究进展 [J]. 软件学报, 2015, 26 (1): 62-81.

[8] 马文婷. 基于 OpenFlow 的 SDN 控制器关键技术研究 [D]. 北京：北京邮电大学, 2015.

[9] 池亚平，王全民. 基于 USBkey 的可信平台模块的研究与仿真设计 [J]. 北京电子科技学院学报, 2007, 15 (4): 13-15.

[10] 张焕国，陈璐，张立强. 可信网络连接研究 [J]. 计算机学报, 2010, 33 (4): 706-717.

[11] 冯登国，秦宇，汪丹，等. 可信计算技术研究 [J]. 计算机研究与发展, 2011, 48 (8): 1332-1349.

[12] 温博为. 可信计算平台技术应用研究 [D]. 西安：陕西师范大学, 2013.

[13] 王晓明. 可信网络远程认证的相关研究 [D]. 济南：山东大学, 2015.

[14] 罗洪达，董增寿，杨威. 基于 TPM 仿真器的可信计算实验平台设计 [J]. 太原科技大学学报, 2013, 34 (5): 337-341.